Chapter 6
Numerical Methods

6.1 Introduction

To solve a scientific problem, we have two methods in a broad sense. They are the ***Analytical method*** and ***Numerical method***.

The traditional and widespread method is the analytical method. In this method first step is to find a formula to calculate the desired dependent value in terms of many other parameters.

$y = f(i, p, f)$

Whereas

i- Independent variables controlling the system directly.

p- Parameters controlling the system indirectly (Reflection parameters).

f- Forcing functions (External influence).

A leak-proof calculation of the answer with this function will yield a highly complicated formula. So, to get a simple dependency function to fit our need with the desired accuracy, we must neglect some parameters. This will yield some errors depending on the neglected parameters and their influence on the system.

The numerical method is a technique by which we can solve a mathematical problem by continues arithmetic operations on observed data. In this method, we can calculate the final answer with desired precision from the observed data sets. The major disadvantage of this method is the large numbers of tedious arithmetic calculations which are time-consuming. But with the aid of a computer, it can be solved quickly.

Advantages of numerical methods

- Numerical methods are extremely powerful problem-solving tools from real data.
- It is capable of handling large systems of equations, complicated geometries, etc. that are often impossible to solve analytically.
- Numerical methods are efficient and accurate than the analytical method for problem-solving with computers.
- Numerical methods are done by reducing higher mathematics to basic arithmetic operations and iterations without neglecting the controlling parameters.

6.2 Curve fitting

A most common problem in scientific experiments is to estimate the value of a dependent variable y for a value of x from a set of date pints x_i&y_i. To solve this we have to find a function f(x) to satisfy the maximum number of available data points. This is known as ***curve fitting***. The function may be for a curve or a straight line. Out of many methods, the least-square approximation is the most popular method.

6.2.1 Least Square Approximation

In the least square approximation method, we are not insisting to pass the function to all individual data points. Instead of that, we will find a single curve or straight line to represent the general trend of the data. This can be performed in a better way by some numerical techniques along with Python.

Let we have some data point $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots (x_n, y_n)$. Then there will be a mathematical equation $y = f(x)$ satisfying all the observed data points. For an observed data point x, the value of $f(x)$ must be equal to observed data point y. The deviation of y from f(x) is the error.

Error in one reading $= y - f(x)$

By least square approximation total error in the analysis

$$S=[y_1 - f(x_1)]^2 + [y_2 - f(x_2)]^2 + \ldots + [y_n - f(x_n)]^2$$

$$S = \sum_{1}^{n} [y_i - f(x_n)]^2 \ldots \ldots \quad (6.1)$$

To get an accurate answer, S must be minimum. For that we can construct a polynomial of any degree depends on desired accuracy level.

6.2.2 Fitting of a straight line

Let $y = mx + c$ is the equation of a straight line which is the best fit. Substituting in equation (6.1)

$$S = \sum_{1}^{n} [y_i - (mx_i + c)]^2$$

For the minimum error

$$\frac{\partial S}{\partial c} = 2 \sum_{1}^{n} [y_i - (mx_i + c)]^2 = 0$$

$$\frac{\partial S}{\partial m} = 2 \sum_{1}^{n} x_i [y_i - (mx_i + c)]^2 = 0$$

On simplification

$$nc + m\sum_{1}^{n} x_i = m\sum_{1}^{n} y_i$$

$$c\sum_{1}^{n} x_i + m\sum_{1}^{n} x_i^2 = \sum_{1}^{n} x_i y_i$$

These are called ***normal equations***, which can be solved to find m and c. Then,

$$m = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x^2{}_i - \sum x_i \sum x_i}$$

$$c = \frac{\sum y_i \sum x^2{}_i - \sum x_i \sum x_i y_i}{n\sum x^2{}_i - \sum x_i \sum x_i}$$

Example 1:

Find the equation of the best fit straight line for the following data points.

X	1	2	4	5	6	8	9
Y	2	5	7	10	12	15	19

Answer:

Serial No	x	v	x^2	xv
1	1	2	1	2
2	2	5	4	10
3	4	7	16	28
4	5	10	25	50
5	6	12	36	72
6	8	15	64	120
7	9	19	81	171
Sum	**35**	**70**	**227**	**453**

Then

$$m = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x^2{}_i - \sum x_i \sum x_i} = \frac{7 \times 453 - 35 \times 70}{7 \times 227 - 35 \times 35} = 1.98$$

$$c = \frac{\sum y_i \sum x^2{}_i - \sum x_i \sum x_i y_i}{n\sum x^2{}_i - \sum x_i \sum x_i} = \frac{70 \times 227 - 35 \times 453}{7 \times 227 - 35 \times 35} = 0.096$$

So the equation of the line is y=1.98x + 0.096

Python code – C.6.1

```
#Programme for straight line fit by least square approximation
```



```
sumx=0
sumy=0
sumxy=0
sumxx=0
n=input('Enter the number of data pairs ')
n=int(n)
for i in range(1,n+1):
    x=input('Enter the X value ')
    x=float(x)
    y=input('Enter the Y value ')
    y=float(y)
    sumx=sumx+x
    sumy=sumy+y
    sumxy=sumxy+x*y
    sumxx=sumxx+x*x
denom=n*sumxx-sumx*sumx
m=(n*sumxy-sumx*sumy)/denom
c=(sumy*sumxx-sumx*sumxy)/denom
print ('The equation of the straight line is y = %7.4f x + %7.4f'%(m,c))
print ('Thank you')
#end
```

Output

```
Enter the number of data pairs  7
Enter the X value  1
Enter the Y value  2
Enter the X value  2
Enter the Y value  5
Enter the X value  4
Enter the Y value  7
Enter the X value  5
Enter the Y value  10
Enter the X value  6
Enter the Y value  12
```

Enter the X value 8
Enter the Y value 15
Enter the X value 9
Enter the Y value 19
The equation of the straight line is y = 1.9808 x + 0.0962
Thank you

6.3 Interpolation

Let us have some data points based on real-time observations. If we do not know the exact relationship between x and y, interpolation can be used to find a polynomial $y = f(x)$ to find the y value for the given x value. If x lies within the range of observed data points, the method is called ***interpolation***. If the x value is outside the range of observed data points, the method is called ***extrapolation***.

6.3.1 Finite difference operator

Let we have some data point $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots (x_n, y_n)$ in which $x_1, x_2, x_3, \ldots x_n$ are equally spaced. ***The finite difference method*** is a method of determination of y value corresponding to a given x value within the range of data. There are 3 types of finite-difference. They are,

- Forward difference
- Central difference
- Backward difference

The operators associated with these methods are generally termed as finite difference operators.

6.3.2 Forward difference operator

For a set of data points $(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$

$$\Delta y_0 = y_1 - y_0$$
$$\Delta y_1 = y_2 - y_1$$
$$\Delta y_2 = y_3 - y_2 \quad \text{So on.}$$

Similarly

$$\Delta^2 y_0 = \Delta y_1 - \Delta y_0$$
$$= (y_2 - y_1) - (y_1 - y_0)$$
$$= y_2 - 2y_1 + y_0$$
$$\Delta^2 y_1 = y_3 - 2y_2 + y_1$$

In the same way, we can do the higher differences.

$$\Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 y_0 \quad \text{So on.}$$

Δ is called the first forward differential operator.

Δ^2 is called second forward differential operator.

Δ^3 is called third forward differential operator. So on.

6.3.3 Backward difference operator

For a set of data points $(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$

$$\nabla y_1 = y_1 - y_0$$
$$\nabla y_2 = y_2 - y_1$$
$$\nabla y_3 = y_3 - y_2 \quad \text{So on.}$$

Similarly

$$\nabla^2 y_2 = \nabla y_2 - \nabla y_1$$
$$\nabla^3 y_3 = \nabla^2 y_3 - \nabla^2 y_2 \quad \text{So on.}$$

∇ is called the first backward differential operator.

∇^2 is called second backward differential operator

∇^3 is called the third backward differential operator. So on.

6.3.4 Central difference operator

For a set of data points $(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$

$$\delta y_{1/2} = y_1 - y_0$$
$$\delta y_{3/2} = y_2 - y_1$$
$$\delta y_{5/2} = y_3 - y_2 \quad \text{So on.}$$

Similarly

$$\delta^2 y_1 = \delta y_{3/2} - \delta y_{1/2}$$
$$\delta^3 y_{3/2} = \delta^2 y_2 - \delta^2 y_1 \quad \text{So on.}$$

δ is called the first central differential operator.

δ^2 is called the first central differential operator.

δ^3 is called the third central differential operator. So on.

While making a difference table if n^{th} differential operator is a constant, then the higher differential operators will vanish. If so, that tabulated values can be represented by a n^{th} degree polynomial with full accuracy. A table with given data points and the higher-order differential operators is known as a ***difference table***. Difference table is the basis for interpolation and numerical differentiation.

Example 2:

Construct a forward difference table for the following data

x	0.25	0.26	0.27	0.28	0.29
y	0.2474	0.2571	0.2667	0.2764	0.2860

Answer:

x	y	Δ	Δ^2	Δ^3	Δ^4
0.25	0.2474				
		0.0097			
0.26	0.2571		-0.0001		
		0.0096		+0.0002	
0.27	0.2667		+0.0001		-0.0004
		0.0097		-0.0002	
0.28	0.2764		-0.0001		
		0.0096			
0.29	0.2860				

Python code for difference table– C.6.2

```
# Difference table
xdata=[]
ydata=[]
n=input('Enter the number of data sets ')
n=int(n)
for i in range(0,n):
    print ('Enter x value of data pair ', i+1, end='  ')
    xi=input()
    print ('Enter y value of data pair ', i+1, end='  ')
    yi=input()
    xi=float(xi)
    yi=float(yi)
    xdata.append(xi)
    ydata.append(yi)
z=[0]
d1=z*n
d2=z*n
d3=z*n
d4=z*n
d5=z*n
print(' FORWARD DIFFERENCE TABLE ')
print ('x      ',sep='  ', end='  ')
for i in range(0,n):
    print ('%8.5f'%(xdata[i]),sep='  ', end='  ')
print ('\ny      ',sep='  ', end='  ')
for i in range(0,n):
    print ('%8.5f'%(ydata[i]),sep='  ', end='  ')
print ('\nDif-1',sep='  ', end='  ')
for i in range(0,n-1):
    d1[i]=ydata[i+1]-ydata[i]
    print ('%8.5f'%(d1[i]),sep='  ', end='  ')
print ('\nDif-2',sep='  ', end='  ')
for i in range(0,n-2):
    d2[i]=d1[i+1] - d1[i]
    print ('%8.5f'%(d2[i]),sep='  ', end='  ')
print ('\nDif-3',sep='  ', end='  ')
for i in range(0,n-3):
    d3[i]=d2[i+1] - d2[i]
    print ('%8.5f'%(d3[i]),sep='  ', end='  ')
print ('\nDif-4',sep='  ', end='  ')
for i in range(0,n-4):
    d4[i]=d3[i+1] - d3[i]
    print ('%8.5f'%(d4[i]),sep='  ', end='  ')
print ('\nDif-5',sep='  ', end='  ')
for i in range(0,n-5):
    d5[i]=d4[i+1] - d4[i]
    print ('%8.5f'%(d5[i]),sep='  ', end='  ')
# End
```

Output

```
Enter the number of data sets 5
```

Enter x value of data pair 1 0.25
Enter y value of data pair 1 0.2474
Enter x value of data pair 2 0.26
Enter y value of data pair 2 0.2571
Enter x value of data pair 3 0.27
Enter y value of data pair 3 0.2667
Enter x value of data pair 4 0.28
Enter y value of data pair 4 0.2764
Enter x value of data pair 5 0.29
Enter y value of data pair 5 0.2860

FORWARD DIFFERENCE TABLE

x	*0.25000*	*0.26000*	*0.27000*	*0.28000*	*0.29000*
y	*0.24740*	*0.25710*	*0.26670*	*0.27640*	*0.28600*
Dif-1	*0.00970*	*0.00960*	*0.00970*	*0.00960*	
Dif-2	*-0.00010*	*0.00010*	*-0.00010*		
Dif-3	*0.00020*	*-0.00020*			
Dif-4	*-0.00040*				
Dif-5					

6.4 Newton's formula for interpolation

Consider a set of data points (x_0, y_0), (x_1, y_1), (x_2, y_2), . . . (x_n, y_n) in which $x_1, x_2, x_3, \ldots x_n$ are equally spaced. Required to find $y_n(x)$, which is a n^{th} degree polynomial such that, $y=y_n(x)$ for all tabulated values.

From the properties of a n^{th} degree polynomial, we can assume that the equation must be in the form

$$y_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + a_3(x-x_0)(x-x_1)(x-x_2) \quad (6.2)$$

Since the points are equidistant

$x_1 = x_0 + h$

$x_2 = x_0 + 2h$

$x_3 = x_0 + 3h$ So on

Then we will get

$x_1 - x_0 = h$

$x_2 - x_0 = 2h$

$x_2 - x_0 = 3h$ So on

To satisfy the first set of data points

$y_0 = y_n(x_0)$

Then equation (6.2) become

$y_0 = a_0$

For second set of data points

$y_1 = y_n(x_1)$ (6.3)

Substituting for a_0 from (6.3) and $y_n(x_1)$ from (6.2)

$y_1 = y_0 + a_1(x_1 - x_0) + a_2(x_1 - x_0)(x_1 - x_1) \ldots$

$y_1 = y_0 + a_1 h$ (6.4)

$$a_1 = \frac{y_1 - y_0}{h} = \frac{\Delta y_0}{h}$$

Similarly

$$a_2 = \frac{\Delta^2 y_0}{2!\,h^2}$$

$$a_3 = \frac{\Delta^3 y_0}{3!\,h^2}$$

Substituting for a_0, a_1, a_2, etc in (6.2)

$$y_n(x) = y_0 + \frac{(x-x_0)\Delta y_0}{h} + \frac{(x-x_0)(x-x_1)\Delta^2 y_0}{2!\,h^2} + \frac{(x-x_0)(x-x_1)(x-x_2)\Delta^3 y_0}{3!\,h^3} + \ldots$$

If $\dfrac{(x-x_0)}{h} = p$

$$y_n(x) = y_0 + p\Delta y_0 + \frac{p(p-1)\Delta^2 y_0}{2!} + \frac{p(p-1)(p-2)\Delta^3 y_0}{3!} + \ldots \quad (6.5)$$

This is the ***Newton's forward difference interpolation*** formula.

Example 3:

Given the following table of readings.

x	0.1	0.2	0.3	0.4
y	1.005	1.020	1.045	1.081

Construct a difference table and find the value of y corresponding to x=0.16

Answer:

x	y	Δ	Δ^2	Δ^3
0.1	1.005			
		0.015		
0.2	1.020		0.010	
		0.025		0.001
0.3	1.045		0.011	
		0.036		
0.4	1.081			

Given x=0.16, h=0.1

$$p = \frac{(x - x_0)}{h} = \frac{(0.16 - 0.1)}{0.1} = 0.6$$

$$y_n(x) = y_0 + p\Delta y_0 + \frac{p(p-1)\Delta^2 y_0}{2!} + \frac{p(p-1)\Delta^3 y_0}{3!} + \ldots$$

$y = 1.005 + 0.06 \times 0.0.015 + 0.6 \times (0.6 - 1) \times 0.010/2 +$
$0.6 \times (0.6 - 1) \times (0.06 - 2) \times 0.001/6$
$\therefore y = 1.012856$

Python code for interpolation– C.6.3

```
#code for Newton's interpolation
xdata=[]
ydata=[]
n=input('Enter the number of data sets  ')
n=int(n)
for i in range(0,n):
    print ('Enter the data pair', i+1,' separated by space', )
    xi,yi=input ().split()
    xi=float(xi)
    yi=float(yi)
    xdata.append(xi)
    ydata.append(yi)
x=input('Enter the data point for interpolation ')
x=float(x)
h=xdata[1]-xdata[0]
```



```
h=float(h)
p=(x-xdata[0])/h
# To make lists with n zero elements,
z=[0]
d1=z*n
d2=z*n
d3=z*n
d4=z*n
d5=z*n
# To enter the difference
for i in range(0,n-1):
    d1[i]=ydata[i+1]-ydata[i]
for i in range(0,n-2):
    d2[i]=d1[i+1] - d1[i]
for i in range(0,n-3):
    d3[i]=d2[i+1] - d2[i]
for i in range(0,n-4):
    d4[i]=d3[i+1] - d3[i]
for i in range(0,n-5):
    d5[i]=d4[i+1] - d4[i]
#Calculation
c1=p
c2=c1*(p-1)/2
c3=c2*(p-2)/3
c4=c3*(p-3)/4
c5=c4*(p-4)/5
y=ydata[0]+c1*d1[0]+c2*d2[0]+c3*d3[0]+c4*d4[0]+c5*d5[0]
print ('The interpolated value for %f is %f' %(x,y))
#end
```

Output

```
Enter the number of data sets  4
Enter the data pair 1  separated by space
0.1 1.005
```

Enter the data pair 2 separated by space
0.2 1.020
Enter the data pair 3 separated by space
0.3 1.045
Enter the data pair 4 separated by space
0.4 1.081
Enter the data point for interpolation 0.16
The interpolated value for 0.160000 is 1.012856

Example 4:

Estimate the population in lakhs in a city in the year 1955 based on the following data with the accuracy of the fifth degree of the polynomial.

Year	1951	1961	1971	1981	1991	2001	2011
Population	35	42	58	84	120	165	220

Answer:

x	y	Δ	Δ^2	Δ^3	Δ^4	Δ^5
1951	35					
		7				
1961	42		9			
		16		1		
1971	58		10		-1	
		26		0		0
1981	84		10		-1	
		36		-1		3
1991	120		9		2	
		45		1		
2001	165		10			
		55				
2011	220					

Given x=1955, h=1961-1951=10

$$p = \frac{(x - x_0)}{h} = \frac{(1955 - 1951)}{10} = 0.4$$

$$y_n(x) = y_0 + p\Delta y_0 + \frac{p(p-1)\Delta^2 y_0}{2!} + \frac{p(p-1)\Delta^3 y_0}{3!} + \cdots$$

$$y = 35 + (0.4 \times 7) - (0.12 \times 9) + (0.064 \times 1) - (0.0416 \times -1) + (0.02995 \times 0)$$

$$\therefore y = 36.826$$

6.5 Numerical Differentiation-First derivative

This is a method of finding the deferential at a point x of a polynomial represented by a set of tabulated data points without knowing the function explicitly. For this, first, construct the difference table. By Newton's forward difference interpolation formula from equation (6.5)

$$y_n(x) = y_0 + p\Delta y_0 + \frac{p(p-1)\Delta^2 y_0}{2} + \frac{p(p-1)(p-2)\Delta^3 y_0}{3!} + \cdots$$

$$y_n(x) = y_0 + p\Delta y_0 + \frac{(p^2-p)\Delta^2 y_0}{2} + \frac{(p^3-3p^2+2p)\Delta^3 y_0}{6} + \cdots$$

Where $p = \frac{(x - x_0)}{h}$

Then $\frac{dp}{dx} = \frac{1}{h}$ (6.6)

We know $\frac{dy}{dx} = \frac{dy}{dp} \times \frac{dp}{dx}$

Sub from (6.5) & (6.6)

$$\frac{dy}{dx} = \left[\Delta y_0 + \frac{(2p-1)\Delta^2 y_0}{2} + \frac{(3p^2-6p+2)\Delta^3 y_0}{6} + \cdots\right] \times \frac{1}{h} \quad (6.7)$$

This is for any non-tabular point x within the range of given table values. That means $x_0 > x > x_n$. On the other hand, if x is a data point itself in the table, equation (4.7) can be simplified further. For that, let us set x as x_0.

Then p=0. Equation (6.7) become

$$\left[\frac{dy}{dx}\right]_{x=x_0} = \left[\Delta y_0 - \frac{\Delta^2 y_0}{2} + \frac{\Delta^3 y_0}{3} + \cdots\right] \times \frac{1}{h} \quad (6.8)$$

Example 5:

Estimate $\frac{dy}{dx}$ at x=0.26 from the following table

x	0.25	0.26	0.27	0.28	0.29
y	0.2474	0.2571	0.2667	0.2764	0.2860

Answer:

x	y	Δ	Δ^2	Δ^3	Δ^4
0.25	0.2474				
		0.0097			
0.26	0.2571		-0.0001		
		0.0096		+0.0002	
0.27	0.2667		+0.0001		-0.0004
		0.0097		-0.0002	
0.28	0.2764		-0.0001		
		0.0096			
0.29	0.2860				

Since x = 0.26 is a tabulated value, reset 0.026 as x_0. Then table changes to

x	y	Δ	Δ^2	Δ^3	Δ^4
0.26	0.2571				
		0.0096			
0.27	0.2667		+0.0001		
		0.0097		-0.0002	
0.28	0.2764		-0.0001		
		0.0096			
0.29	0.2860				

$$\left[\frac{dy}{dx}\right]_{x=x_0} = \left[\Delta y_0 - \frac{\Delta^2 y_0}{2} + \frac{\Delta^3 y_0}{3}\right] \times \frac{1}{h}$$

$$= \left[0.0096 - \frac{0.0001}{2} + \frac{-0.0002}{3}\right] \times \frac{1}{0.01} = 0.9483$$

Example 6:

The following table gives the variation of the velocity of an object at various points of time. Find the acceleration at t=1.2 second.

Time(s)	1.0	1.4	1.8	2.2
Velocity(m/sec)	2.7183	4.0552	6.4096	9.0250

Answer:

Required to find the first differential of velocity at t=1.2. For that construct, the difference table.

x	y	Δ	Δ^2	Δ^3
1.0	2.7183			
		1.3369		
1.4	4.0552		1.0175	
		2.3544		-0.7565
1.8	6.4096		0.261	
		2.6154		
2.2	9.0250			

$h = 1.4 - 1.0 = 0.4$

$$p = \frac{x - x_0}{h} = \frac{1.2 - 1.0}{0.4} = 0.5$$

$$\frac{dy}{dx} = \left[\Delta y_0 + \frac{(2p-1)\Delta^2 y_0}{2} + \frac{(3p^2 - 6p + 2)\Delta^3 y_0}{6}\right]\frac{1}{h}$$

$$= \left[1.3369 + \frac{0 \times 1.0175}{2} + \frac{-0.25 \times -0.7565}{6}\right]\frac{1}{0.4}$$

$$= [1.3369 + 0 + 0.031]\frac{1}{0.4} = 3.4210 \text{ ms}^{-2}$$

Python code for First order differentiation– C.6.4

```
xdata=[]
ydata=[]
n=input('Enter the number of data sets ')
n=int(n)
for i in range(0,n):
    xdata=[]
    ydata=[]
for i in range(0,n):
    print ('Enter the data pair', i+1,' separated by space', end=' ')
    xi,yi=input().split()
    xi=float(xi)
    yi=float(yi)
    xdata.append(xi)
    ydata.append(yi)
x=input('Enter the data point for differentiation ')
x=float(x)
```

```
h=float(xdata[1]-xdata[0])
p=float((x-xdata[0])/h)
z=[0.0]
d1=z*n
d2=z*n
d3=z*n
d4=z*n
for i in range(0,n-1):
    d1[i]=ydata[i+1]-ydata[i]
for i in range(0,n-2):
    d2[i]=d1[i+1] - d1[i]
for i in range(0,n-3):
    d3[i]=d2[i+1] - d2[i]
for i in range(0,n-4):
    d4[i]=d3[i+1] - d3[i]
c2=(2.0*p-1)/2
c3=(3.0*p*p-6*p+2)/6
c4=(3.0*p*p*p-18*p*p+22*p-6)/24
diff=(d1[0]+c2*d2[0]+c3*d3[0]+c4*d4[0])/h
print('The first differential is %f' %(diff))
print('Thank You')
#End
```

Output

```
Enter the number of data sets  4
Enter the data pair 1 separated by space  1.0 2.7183
Enter the data pair 2 separated by space  1.4 4.0552
Enter the data pair 3 separated by space  1.8 6.4096
Enter the data pair 4 separated by space  2.2 9.0250
Enter the data point for differentiation  1.2
The first differential is 3.421052
Thank You
```


6.6 Numerical Differentiation-Second derivative

From eq. 6.7

$$\frac{dy}{dx} = \left[\Delta y_0 + \frac{(2p-1)\Delta^2 y_0}{2} + \frac{(3p^2-6p+2)\Delta^3 y_0}{6} + \cdots\right] \times \frac{1}{h}$$

$$\frac{d^2y}{dx^2} = \frac{d}{dx}\left(\frac{dy}{dx}\right) = \frac{d}{dp}\left(\frac{dy}{dx}\right) \times \frac{dp}{dx}$$

From eq. 6.7

$$\frac{dy}{dx} = \left[\Delta y_0 + \frac{(2p-1)\Delta^2 y_0}{2} + \frac{(3p^2-6p+2)\Delta^3 y_0}{6} + \cdots\right] \times \frac{1}{h}$$

And from eq. 6.6, $\frac{dp}{dx} = \frac{1}{h}$

Substituting,

$$\frac{d^2y}{dx^2} = \frac{d}{dp}\left(\left[\Delta y_0 + \frac{(2p-1)\Delta^2 y_0}{2} + \frac{(3p^2-6p+2)\Delta^3 y_0}{6} + \cdots\right] \times \frac{1}{h}\right) \times \frac{1}{h}$$

$$= \frac{d}{dp}\left(\Delta y_0 + \frac{(2p-1)\Delta^2 y_0}{2} + \frac{(3p^2-6p+2)\Delta^3 y_0}{6} + \frac{(4p^3-18p^2+22p-6)\Delta^4 y_0}{24} \cdots\right) \times \frac{1}{h^2}$$

$$= \left[0 + \frac{2\,\Delta^2 y_0}{2} + \frac{(6p-6)\Delta^3 y_0}{6} + \frac{(12p^2-36p+22)\Delta^4 y_0}{24} \cdots\right] \times \frac{1}{h^2}$$

$$\therefore \frac{d^2y}{dx^2} = \left[\Delta^2 y_0 + (p-1)\Delta^3 y_0 + \frac{(12p^2-36p+22)\Delta^4 y_0}{24} \cdots\right] \times \frac{1}{h^2}$$

Example 7:

Estimate $\frac{dy}{dx}$ at x=0.255 from the following table

x	0.25	0.26	0.27	0.28	0.29
y	0.2474	0.2571	0.2667	0.2764	0.2860

Answer

x	y	Δ	Δ^2	Δ^3	Δ^4
0.25	0.2474				
		0.0097			

x	y	Δ	Δ^2	Δ^3	Δ^4
0.26	0.2571		-0.0001		
		0.0096		+0.0002	
0.27	0.2667		+0.0001		-0.0004
		0.0097		-0.0002	
0.28	0.2764		-0.0001		
		0.0096			
0.29	0.2860				

$h = 0.26 - 0.25 = 0.01$

$$p = \frac{x - x_0}{h} = \frac{0.255 - 0.25}{0.01} = 0.5$$

$$\frac{d^2y}{dx^2} = \left[\Delta^2 y_0 + (p-1)\Delta^3 y_0 + \frac{(12p^2 - 36p + 22)\Delta^4 y_0}{24} \ldots\right] \times \frac{1}{h^2}$$

=-0.0001 + (0.5-1)x0.0002 + (

$$= \left[-0.0001 + (0.5-1) \times 0.0002 + \left(\frac{12 \times 0.5^2 - 36 \times 0.5 + 22}{24} \times -0.0004\right)\right] \times \frac{1}{0.01^2}$$

= -3.16667

Python code for First order differentiation– C.6.5

```
#code for second derivative
xdata=[]
ydata=[]
n=input('Enter the number of data sets ')
n=int(n)
for i in range(0,n):
  xdata=[]
  ydata=[]
for i in range(0,n):
  print ('Enter the data pair', i+1,' separated by space', end=' ')
  xi,yi=input().split()
  xi=float(xi)
```



```
  yi=float(yi)
  xdata.append(xi)
  ydata.append(yi)
x=input('Enter the data point for differentiation ')
x=float(x)
h=float(xdata[1]-xdata[0])
p=float((x-xdata[0])/h)
z=[0.0]
d1=z*n
d2=z*n
d3=z*n
d4=z*n
for i in range(0,n-1):
  d1[i]=ydata[i+1]-ydata[i]
for i in range(0,n-2):
  d2[i]=d1[i+1] - d1[i]
for i in range(0,n-3):
  d3[i]=d2[i+1] - d2[i]
for i in range(0,n-4):
  d4[i]=d3[i+1] - d3[i]
c3=(p-1.0)
c4=(12.0*p*p-36*p+22)/24
diff=((d2[0]+c3*d3[0]+c4*d4[0]))/(h*h)
print('The second differential is %f' %(diff))
print('Thank You')
#End
```

Output

```
Enter the number of data sets  5
Enter the data pair 1 separated by space  0.25 0.2474
Enter the data pair 2 separated by space  0.26 0.2571
Enter the data pair 3 separated by space  0.27 0.2667
Enter the data pair 4 separated by space  0.28 0.2764
Enter the data pair 5 separated by space  0.29 0.2860
```

Enter the data point for differentiation 0.255
The second differential is -3.166667
Thank You

6.7 Numerical Integration by Newton – Cotes quadrature formulas

The Newton-Cotes formulas are the most used numerical integration methods. In this, the integration of a complicated function is replaced by many Polynomials across the integration interval. To find the fitting polynomials Lagrange interpolating polynomials can be used. The resulting formulas are called Newton-Cotes Quadrature Formulas. The integration of the original function can then be obtained by summing up all such polynomials whose "areas" are calculated by many methods. For accuracy, the entire area under the curve can be split into several small strips. The x value of such strips is called nodal points. Along with these areas, we will use some coefficients called weighting coefficients.

Depends on the weighting coefficients and the method of summing up of the areas there are many methods like trapezoidal method, Simpson's Rule, Simpson's 3/8 Rule, Boole's Rule, etc.

Nodal points can be obtained by incrementing the initial value of x with a small value called step size. This method of integration can be done by connecting polynomials or without the polynomials. If we do not know the exact polynomial equations, we can do the integration with observed data in regular intervals. If we have the limits of integration, we must use closed Newton – Cotes quadrature and if the limits are not known, we have to use open Newton – Cotes quadrature

Let we have some data point (x_1, y_1), (x_2, y_2), (x_3, y_3), . . . (x_n, y_n) connected by a function $y=f(x)$, where $f(x)$ is not known explicitly. Numerical integration is a method to compute the definite integral of this function by using the observed data points.

$$I = \int_a^b y\,dx$$

Divide the interval [a,b] into n subintervals of equal width, such that $x_0=a$ and $x_n=b$.

$x_1 = x_0 + h$
$x_2 = x_0 + 2h$
$x_3 = x_0 + 3h$
. . . . . .
$x_n = x_0 + nh$

Then the definite integral becomes

$$I = \int_{x_0}^{x_n} y\,dx$$

Substituting for y from Newton's forward difference interpolation formula as in equation (6.5)

$$I = \int_{x_0}^{x_n} \left[y_0 + p\Delta y_0 + \frac{p(p-1)\Delta^2 y_0}{2} + \frac{p(p-1)(p-2)\Delta^3 y_0}{6} + \cdots \right] dx$$

But

$p = \frac{(x - x_0)}{h}$

$x = x_0 + ph$

$dx = h\,dp$

$x_0 \to 0$

$x_n \to n$

Substituting

$$I = h \int_0^n \left[y_0 + p\Delta y_0 + \frac{p(p-1)\Delta^2 y_0}{2} + \frac{p(p-1)(p-2)\Delta^3 y_0}{6} + \cdots \right] dp$$

$$I = h \int_0^n \left[y_0 + p\Delta y_0 + \frac{(p^2-p)\Delta^2 y_0}{2} + \frac{(p^3 - 3p^2 + 2p)\Delta^3 y_0}{6} + \cdots \right] dp$$

$$I = h \left[n\,y_0 + \frac{n^2}{2}\Delta y_0 + \frac{\left(\frac{n^3}{3} - \frac{n^2}{2}\right)\Delta^2 y_0}{2} + \frac{\left(\frac{n^4}{4} - \frac{3n^3}{3} + \frac{2n^2}{2}\right)\Delta^3 y_0}{6} + \cdots \right]$$

$$I = h \left[n\,y_0 + \frac{n^2}{2}\Delta y_0 + \frac{(2n^3 - 3n^2)\Delta^2 y_0}{12} + \frac{(n^4 - 4n^3 - 4n^2)\Delta^3 y_0}{24} + \cdots \right] \quad (6.9)$$

This is a general formula to find the polynomial for numerical integration. Different methods can be derived from this fundamental equation by setting different values for n.

6.8 Trapezoidal Method

In this method, the interval [a,b] is divided into n sub-elements. Since each element is analysing separately, n=1

That means,

$$I = \int_a^b y\,dx = \int_{x_0}^{x_1} y\,dx + \int_{x_1}^{x_2} y\,dx + \ldots + \int_{x_{n-1}}^{x_n} y\,dx \qquad (6.10)$$

If n=1, in each subinterval, $\Delta^2 y_0$ and higher terms will vanish from the difference table. Substituting these conditions in equation (6.9)

$$\int_{x_0}^{x_1} y\,dx = h\left[y_0 + \frac{1}{2}\Delta y_0\right]$$

$$= h\left[y_0 + \frac{1}{2}(y_1 - y_0)\right]$$

$$= \frac{h}{2}[y_0 + y_1]$$

Similarly

$$\int_{x_1}^{x_2} y\,dx = \frac{h}{2}[y_1 + y_2]$$

So on

Substituting in (6.10)

$$\int_a^b y\,dx = \frac{h}{2}[(y_0 + y_1) + (y_1 + y_2) + \cdots + y_{n-1} + y_n]$$

$$= \frac{h}{2}[y_0 + 2y_1 + 2y_2 + \cdots + 2y_{n-1} + y_n]$$

$$= \frac{h}{2}[y_0 + 2(y_1 + y_2 + \cdots + y_{n-1}) + y_n]$$

This is known as the trapezoidal rule.

Python code for Trapezoidal rule

1. Numerical integration with data points– C.6.6

```
#Trapezoidal with data
xdata=[]
ydata=[]
tot=0.0
n=input('Enter the number of data sets   ')
n=int(n)
```



```
for i in range(0,n):
    print ('Enter the data pair', i+1, ' separated by space', end=' ')
    xi,yi=input().split()
    xi,yi=float(xi),float(yi)
    xdata.append(xi)
    ydata.append(yi)
h=xdata[1]-xdata[0]
h=float(h)
tot=ydata[0]+ydata[n-1]
for i in range (1,n-1):
    tot=tot+2*ydata[i]
integ=tot*h/2.0
print ('The integral within the given limit is ', integ)
Thank You
#End
```

Output

```
Enter the number of data sets  6
Enter the data pair 1 separated by space  0 2
Enter the data pair 2 separated by space  0.2 2.04
Enter the data pair 3 separated by space  0.4 2.16
Enter the data pair 4 separated by space  0.6 2.36
Enter the data pair 5 separated by space  0.8 2.64
Enter the data pair 6 separated by space  1.0 3.00
The integral within the given limit is  =  2.3400
```

2. Numerical integration for known function - C.6.7

```
#Trapezoidal with data
def f(x):
    # Here you can enter the given function
    y = (x*x*x+2*x)/(x*x+2*x)
    return y
a= input('Enter the lower limit ')
b= input('Enter the upper limit ')
h= input('Enter the step size ')
```

```
a,b,h=float(a),float(b),float(h)tot= f(a)+f(b)
x=a+h
while abs(x- b)>0.00001:
    tot=tot+2*f(x)
    x=x+h
integ=tot*h/2.0
print ('The integral within the given limit is = %9.4f' %integ)
#Thank You
#End
```

Output

```
Enter the lower limit 1.0
Enter the upper limit 2.0
Enter the step size 0.2
The integral within the given limit is =   1.2271
```

Example 8:

Using trapezoidal rule with a step size 0.2, find

$$\int_1^2 \frac{x^3 + 2x}{x^2 + 2x}\, dx$$

Answer

x	1.0	1.2	1.4	1.6	1.8	2.0
y	1.000	1.075	1.165	1.267	1.379	1.500

$$\int_a^b y\, dx = \frac{h}{2}[y_0 + 2(y_1 + y_2 + \cdots + y_{n-1}) + y_n]$$

$$= \frac{0.2}{2}[1.000 + 2(1.075 + 1.164 + 1.267 + 1.379) + 1.5]$$

$$\therefore \int ydx = 1.2270$$

Example 9:

From the following table estimate the area bounded by the curve and the x-axis from x=0 to x=1

x	0	0.2	0.4	0.6	0.8	1.0
y	2.00	2.04	2.16	2.36	2.64	3.00

Answer

$$\int_a^b y\, dx = \frac{h}{2}[y_0 + 2(y_1 + y_2 + \cdots + y_{n-1}) + y_n]$$

$$= \frac{0.2}{2}[2.00 + 2(2.04 + 2.16 + 2.36 + 2.64) + 3.00]$$

$$= 2.34$$

$$\therefore Area = 2.34$$

6.9 Simpson's 1/3 Rule

This is a more accurate and popular method of numerical integration in comparison with the trapezoidal method. In this method, the interval [a,b] is divided into n sub-elements. Since two elements are coupled for analysis, n=2

That means equation (6.10) become,

$$I = \int_a^b y\, dx = \int_{x_0}^{x_2} y\, dx + \int_{x_2}^{x_4} y\, dx + \ldots + \int_{x_{n-2}}^{x_n} y\, dx \qquad (6.11)$$

If n=2, in each subinterval, $\Delta^3 y_0$ and higher terms will vanish from the difference table. Substituting these conditions in equation (6.9)

$$\int_{x_0}^{x_2} y\, dx = h\left[2y_0 + 2\Delta y_0 + \frac{1}{3}\Delta^2 y_0\right]$$

Substituting for Δy_0 and $\Delta^2 y_0$ from difference table

$$\int_{x_0}^{x_2} y\, dx = h\left[2y_0 + 2(y_1 - y_0) + \frac{1}{3}(y_2 - 2y_1 + y_0)\right]$$

$$= \frac{h}{3}[6y_0 + 6(y_1 - y_0) + (y_2 - 2y_1 + y_0)]$$

$$= \frac{h}{3}[y_0 + 4y_1 + y_2]$$

Similarly

$$\int_{x_2}^{x_4} y\,dx = \frac{h}{3}[y_2 + 4y_3 + y_4]$$

So on. Finally,

$$\int_{x_{n-2}}^{x_n} y\,dx = \frac{h}{3}[y_{n-2} + 4y_{n-1} + y_n]$$

Substituting in (6.11)

$$I = \int_a^b y\,dx = \frac{h}{3}[y_0 + 4y_1 + y_2] + \frac{h}{3}[y_2 + 4y_3 + y_4] + \cdots + \frac{h}{3}[y_{n-2} + 4y_{n-1} + y_n]$$

$$= \frac{h}{3}[y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n]$$

For non-computer applications, it can be rewritten as

$$I = \frac{h}{3}[y_0 + 4(y_1 + y_3 + y_5 + \cdots + y_{n-1}) + 2(y_2 + y_4 + y_6 + \cdots + y_{n-2}) + y_n] \quad (6.12)$$

Note:

1. If we divide the function into odd number of subintervals the answer will not be accurate, since last segment cannot be paired properly.
2. When number of subintervals increases, accuracy increases.

Python code for Simpson's 1/3 rule with data points- C.6.8

```
#simpson 1/3 rule
xdata=[]
ydata=[]
tot=0.0
n=input('Enter the number of data sets  ')
n=int(n)
for i in range(0,n):
  print ('Enter the data pair', i+1,' separated by space',end=' ')
  xi,yi=input().split()
  xi=float(xi)
```



```
  yi=float(yi)
  xdata.append(xi)
  ydata.append(yi)
h=xdata[1]-xdata[0]
h=float(h)
tot=ydata[0]+ydata[n-1]
for i in range (1,n,2):
  tot=tot+4*ydata[i]
for i in range (2,n-1,2):
  tot=tot+2*ydata[i]
integ= tot*h/3
print ('The value of integral = %9.4f' %integ)
print ('Thank You')
#End
```

Output

```
Enter the number of data sets  7
Enter the data pair 1 separated by space  0 0
Enter the data pair 2 separated by space  2 22
Enter the data pair 3 separated by space  4 30
Enter the data pair 4 separated by space  6 27
Enter the data pair 5 separated by space  8 18
Enter the data pair 6 separated by space  10 7
Enter the data pair 7 separated by space  12 0
The value of integral = 213.3333
Thank You
```

Python code for Simpson's 1/3 rule for known function - C.6.8

```
#simpson 1/3 rule
def f(x):
  # Here you can enter the given function
  y = (x*x*x+2*x)/(x*x+2*x)
  return y
a= input('Enter the lower limit ')
b= input('Enter the upper limit ')
```

```
h= input('Enter the step size ')
a,b,h=float(a),float(b),float(h)
tot= f(a)+f(b)
x=a+h
while abs(x- b)>0.00001:
    tot=tot+4*f(x)
    x=x+h
    if abs(x- b)<0.00001:
        break
    tot=tot+2*f(x)
    x=x+h
integ= tot*h/3
print ('The value of integral = %9.4f' %integ)
print ('Thank You')
#End
```

Output

```
Enter the lower limit 1.0
Enter the upper limit 2.0
Enter the step size 0.1
The value of integral =   1.2261
Thank You
```

Example 10:

Using Simpson's 1/3 rule with a step size 0.1, find

$$\int_1^2 \frac{x^3 + 2x}{x^2 + 2x}\, dx$$

Answer

x	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0
y	1.000	1.035	1.075	1.118	1.165	1.214	1.267	1.322	1.379	1.438	1.500

$$I = \frac{h}{3}[y_0 + 4(y_1 + y_3 + y_5 + \cdots + y_{n-1}) + 2(y_2 + y_4 + y_6 + \cdots + y_{n-2}) + y_n]$$

$$I = \frac{0.1}{3}[1 + 4(1.035 + 1.118 + 1.214 + 1.322 + 1.438) + 2(1.075 + 1.165 + 1.267 + 1.379) + 1.500]$$

$$I = \frac{0.1}{3}[36.78] = 1.226$$

Example 11:

The velocities of a car running on a straight road at intervals of 2 minutes are given below. Find the distance covered using Simpson's rule.

Time	0	2	4	6	8	10	12
velocity	0	22	30	27	18	7	0

Answer

$$I = \frac{h}{3}[y_0 + 4(y_1 + y_3 + y_5) + 2(y_2 + y_4) + y_n]$$

$$= \frac{2}{3}[0 + 4(22 + 27 + 7) + 2(30 + 18) + 0]$$

$$= \frac{2}{3}[320]$$

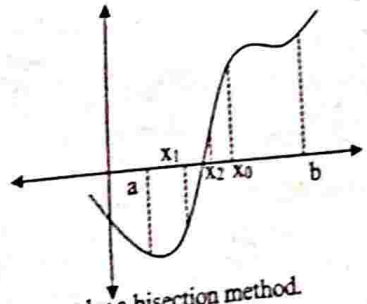
$$\therefore Distance = 213.33$$

6.10 Solution of algebraic and transcendental equations

The determination of the roots of an equation in the form f(x)=0 is the most common problem in science and engineering work. If the function is quadratic or cubic, we have some readymade formulas. If it is a polynomial of a higher degree or a transcendental function, readymade formulas are not available. We have many numerical iteration methods like the bisection method, Newton-Raphson method, etc for such functions. These methods can be used to find the square root, cube root, etc of any number.

6.11 Bisection method

This is a method to find the root of an equation. Even though it is not a faster method, it is the most reliable method in numerical mathematical methods. The root means value of x_0 for which $f(x_0) = 0$. That will be the cutting point of the curve and x-axis.



The following are the different steps to find the root by a bisection method.

Step 1: Let f(x) is the given function. First, assume two values for x as 'a' and 'b'

so that a<b and f(a) & f(b) are opposite in sign. One must be positive and the other must be negative. If our guess is right f(a) X f(b) < 0. Then the actual root is in between a & b. Otherwise, we must make another guess.

Step 2: If f(a) = 0, a is a root. Similarly, if f(b)=0, b is a root. In a computer to get or compare an exact zero value for a float is difficult. Usually, instead of comparing a value with zero, if the modulus of the value is less than a tolerable, very small, error limit, we will treat that number as zero. That means if |f(a)| < e, a will be a root or if |f(b)| < e, b will be a root. The value of e must be very small like 0.00001. It is as per the choice of the user, depending on the accuracy level.

Step 3: If any one of the guess values is not a root, assume a new value for root as $x_0=(a+b)/2$

Step 4: Check whether x_0 is a root as before. If it is, then the problem has been solved. Else find $f(x_0)$. If $f(x_0)$ X f(b) > 0, both are with same sign. Either positive or negative. That means x_0 and b are on the same side of the root point. Then change x_0 as the new value of b. Else change x_0 as new value of a. Again, find new value for root x_0 as $x_1=(a+b)/2$. Then repeat from step 2 onwards, till we are getting the root within the desired error limit. After each iteration, we can see that the calculated root is converging near to the actual root.

As the number of iterations increases, we are very close to the exact and answer. Depends on the accuracy requirement we can stop it at any time. Numerically, these iterations will be tedious, if the value of x_0 is not converging within few steps. But using a computer programme, it is very simple.

Example12:

Find the root of $2x^2 + 3x - 14 = 0$. Do a maximum of 6 iterations only.

Answer:

Guess value: Assume that a=1 and b=4

f(a)=-9 and f(b)=30. Since both are off different signs, a and b can be accepted.

x_0= a+b/2 = (1+4)/2=2.5

Iteration 1: $f(x_0)=f(2.5)$ =2.5

$f(x_0)$ and f(b), both are of same sign, then replace b.

New a=1, new b=2.5, new root x_1= (1+2.5)/2 = 1.75

Iteration 2: $f(x_1)$ =f(1.75) =-2.625

$f(x_0)$ and f(b), both are of different sign, then replace a.

New a=1.75, new b=2.5, new root x_2= (1.75+2.5)/2 = 2.125

Iteration 3: $f(x_2)$ =f(2.125) =1.4062

$f(x_0)$ and f(b), both are of same sign, then replace b.

New a=1.75, new b=2.125, new root x_3= (1.75+2.125)/2 = 1.9375

Iteration 4: $f(x_3)$ =f(1.9375) =-0.6797

$f(x_0)$ and f(b), both are of different sign, then replace a.

New a=1.9375, new b=2.125, new root x_4= (1.9375+2.125)/2 = 2.0312

Iteration 5: $f(x_4)$ =f(2.0312) =0.3451

$f(x_0)$ and f(b), both are of same sign, then replace b.

New a=1.9375, new b=2.0312, new root x_5= (1.9375+2.0312)/2 = 1.9844

Iteration 6: $f(x_5)$ =f(1.9844) =-0.1711

$f(x_0)$ and f(b), both are of different sign, then replace a.

New a=1.9844, new b=2.0312, new root x_6= (1.9844+2.0312)/2 = 2.0078

Iteration 7: $f(x_6)$ =f(2.0078) =0.08592

$f(x_0)$ and f(b), both are of same sign, then replace b.

New a=1.9844, new b=2.0078, new root x_7= (1.9844+2.0078)/2 = 1.9961

The root of the given equation =1.9961 after 7 iterations.

Python code for Bisection method - C.6.9

```
#Bisection
from math import *
import sys
def f(x):
  # Here you can enter the given function
  y = 2*x*x+3*x-14
  return y
a= float(input('Enter the lower guess value '))
b= float(input('Enter the upper guess value '))
while f(a)*f(b)>0:
  print('Your guess value is not proper, try with a new value')
  a= float(input('Enter the lower guess value '))
  b= float(input('Enter the upper guess value '))
err= float(input('Enter the error limit     '))
if abs(f(a))<err :
  print ('The root is = %9.4f' %a)
  sys.exit()
if abs(f(b))<err :
```

```
    print ('The root is = %9.4f' %b)
    sys.exit()
x0=(a+b)/2
counter=0
if abs(f(x0))<=err : print ('The root is = %9.4f' %x0)
while abs(f(x0))>err:
  if f(x0)*f(b)>0 :
    b=x0
  else :
    a=x0
  counter=counter+1
  x0=(a+b)/2.0
  if abs(f(x0))<=err:
    print ('The number of iterations = %4d' %counter)
    print ('The accuracy level       = %9.5f' %err)
    print ('The root is              = %9.5f' %x0)
    break
#End
```

Output

```
Enter the lower guess value 1
Enter the upper guess value 4
Enter the error limit        0.00001
The number of iterations  =  20
The accuracy level        =  0.000010
The root is               =  2.00000
```

6.12 Newton-Raphson method

This is a more accurate method to improve the approximate root obtained by some other methods which are not so accurate. Let x_0 is an approximate or guessed root of the equation f(x) = 0. Then the correct root $x_1 = x_0 + h$ so that $f(x_1) = 0$. We can assume that h is the correction.

Substituting for x_1, $f(x_0 + h) = 0$

Expanding with Tailer series

$$f(x_0) + hf'(x_0) + \frac{h^2}{2!}f''(x_0) + \ldots = 0$$

Neglecting the higher-order terms, we have

$$f(x_0) + hf'(x_0) = 0$$

$$h = -\frac{f(x_0)}{f'(x_0)}$$

So the approximate root x_0 can be converted into a better root x_1 by

$x_1 = x_0 + h$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

If x_1 is not an answer with desired accuracy level, improve it again by

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

In general

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

This can be continued, till we are getting the root with the desired accuracy. This formula is known as the ***Newton-Raphson formula***. This method is widly using to find any root of any number

Let we want to find $\sqrt[3]{12}$. Then,

$x^3 = 12$

$x^3 - 12 = 0$

Solve this equation to find x.

Python code for Newton-Raphson method - C.6.10

```
#Solving equations-Newton Raphson
def y(x):
  y=x*x-3*x+2
  return y
def y1(x):
  y1=2*x-3
  return y1
```

```
x=input('Enter a guess values :  ')
x=float(x)
while abs(y(x))> 0.0000001:
    x=x-y(x)/y1(x)
print ('The solution by Newton Raphson method is %9.4f ' %x)
#end
```

Output

Enter a guess values : 0

The solution by Newton Raphson method is 1.0000

Example 13:

Using the Newton-Raphson method, find the root of

$x^2 - 3x + 2$

Answer:

$f(x) = x^2 - 3x + 2$

$f'(x) = 2x - 3$

Assume the answer x_1 as 0

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 0 - \frac{2}{-3} = 0.6667$$

$$x_3 = 0.6667 - \frac{0.4444}{-1.6667} = 0.9333$$

$$x_4 = 0.9333 - \frac{0.071}{-1.334} = 0.9959$$

$$x_5 = 0.9959 - \frac{0.0041}{-1.0082} = 0.9999$$

$$x_6 = 0.9999 - \frac{0.0001}{-1.0002} = 1.000$$

Since the difference between x_5 and x_6 is negligible (or we can say f(x) is very small), $x_6$is almost close to the answer.

So, the most correct root is 1.000. If we want more accuracy, the iteration can be continued.

$\therefore x = 1.000$

Example 14:

Using the Newton-Raphson method, find $\sqrt[2]{7}$

Answer:

$x^2 = 7$

$x^2 - 7 = 0$

$f(x) = x^2 - 7$

$f'(x) = 2x$

Assume the answer x_1 as 2.5 since $\sqrt[2]{4} = 2$ and $\sqrt[2]{9} = 3$

$f(2.5) \neq 0$

Then the new approximation

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 2.5 - \frac{-0.75}{5} = 2.65$$

$$x_3 = 2.65 - \frac{0,0225}{5.3} = 2.646$$

$$x_4 = 2.646 - \frac{0.00131}{5.292} = 2.6457$$

Since the difference between x_3 and x_4 is negligible, x_4 is almost close to the answer.

$\therefore \sqrt[2]{7} = 2.6457$

6.13 Numerical solutions of differential equations

Many problems in science and engineering can be simplified into some form of differential equation that satisfies the given conditions. There are many numerical methods like the Euler method, Heun's method, Runge-Kutta method, Adam's method, etc to solve the differential equations very easily. Consider a differential equation $y' = f(x, y)$ with initial condition $y(x_0) = y_0$. We want to find y value for a given value of x.

6.14 Euler's Method

Euler's method is the simplest method to solve a differential equation, but with low accuracy. Consider$y' = f(x, y)$. By Tailor's series expansion for y(x) around $x=x_0$

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + \frac{(x - x_0)^2}{2!}y''(x_0) + \cdots$$

Neglecting higher terms

$y(x) = y(x_0) + (x - x_0)y'(x_0)$

$y(x) = y(x_0) + (x - x_0)f(x_0, y_0)$

For any value x_1 very close to x_0 such that, $x_1 = x_0 + h$

$y(x_1) = y(x_0) + (x_1 - x_0)f(x_0, y_0)$

$y(x_1) = y(x_0) + hf(x_0, y_0)$

Similarly

$y(x_2) = y(x_1) + hf(x_1, y_1)$

Extending to higher values

$x_{i+1} = x_i + h$ and $y_{i+1} = y_i + hf(x_i, y_i)$ (6.13)

This formula is called ***Euler's formula***. In this method, starting from an initial data pair, we can reach to the required x value and corresponding f(x) by step-by-step change. In each stage, a new value of y is estimated by using previous values of x and y. Here $f(x_i,y_i) = \frac{dy}{dx}$, which is the slope of function plot at (x_i, y_i).

New value of a variable=Old value + Slope X Step size

$y_{i+1} = y_i$ + slope x h

Python code for Euler's Method - C.6.11

```
#Euler's Method
def f(x,y):
    y=3*x*x+1
    return y
x= float(input('Initial value of x = '))
y= float(input('Initial value of y = '))
xf= float(input('Final value of x   = '))
h= float(input('Step size          = '))
print ('   x- Value          y- Value')
while x <= xf:
    print ('   %6.3f          %6.3f' %(x,y))
    if x + h > xf: break
    y = y + h*f(x,y)
    x = x + h
print ('Integral of the given function = %9.4f%y)
print ('Thank you')
#end
```

Output

```
Initial value of x = 1
Initial value of y = 2
Final value of x   = 2
Step size          = 0.25
  x- Value          y- Value
   1.000             2.000
   1.250             3.000
   1.500             4.422
   1.750             6.359
   2.000             8.906
Integral of the given function =.  8.9062
Thank you
```


Example 15:

$dy/dx = 3x^2 + 1$ is with an initial value, y=2 when x=1. Solve it for x=2 with a step size of 0.25

Answer

Given $f(x) = 3x^2 + 1, x_0 = 1, y_0 = 2$

Step 1 y(1)=2 given

Step 2 $y=y_0+hf(x_0, y_0)$

$y(1.25) = 2 + 0.25(3 \times 1^2 + 1) = 3$

Step 3 $y(1.5) = 3 + 0.25(3 \times 1.25^2 + 1)$

$= 4.4218$

Step 4 $y(1.75) = 4.4218 + 0.25(3 \times 1.5^2 + 1)$

$=6.3593$

Step 5 $y(2) = 6.3593 + 0.25(3 \times 1.75^2 + 1)$

$=8.9062$

$\therefore y(2) = 8.9062$

6.15 Runge-Kutta methods(R-K methods)

Runga–Kutta method refers to a family of one-step methods for the numerical solution of differential equations. Runga-Kutta methods are known by their order. Euler's method is a first-order R-K method. Similarly, we have Second-order R-K, Fourth-order R-K, etc. In the R-K family, as order increases, accuracy also increases.

6.15.1 Second order Runge-Kutta method

Second-order R-K is more accurate than Euler's method. First order R-K is same

as Euler method. Second-order R-K is known as Heun's method.

In the case of Euler's method

$$y_{i+1} = y_i + \text{slope} \times \text{step size}$$

$$y_{i+1} = y_i + mh$$

Here m is the slope of the function at a point.

In Euler's method, we are considering the slope of a single point. That is the reason for the deviation of the answer. Instead of taking the slope at one point, we can measure the slope at different points. The average of this can be used for further calculation. The order of the R-K method is exactly, the number of slopes that we are considering for the calculation of the next step. So, Euler's method is a first-order method since we are considering only one slope for the calculation of next point.

In the second-order method, we must consider the slope at two points. One slope is from the previous point and the other slope form the forthcoming point by extrapolation. The average of these two can be used to find the final solution.

By Euler's equation

$$y_{i+1} = y_i + m_1 h \quad \text{Where } m_1 = f(x_i, y_i)$$

If we are using the slope at a calculated new point, this equation will be modified as,

$$y_{i+1} = y_i + m_2 h \quad \text{Where } m_2 = f(x_{i+1}, y_{i+1})$$

Average slope $m = \dfrac{m_1 + m_2}{2}$

$$m = \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1})}{2}$$

Substituting in Euler's equation

$$Y_{i+1} = y_i + \left(\frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1})}{2}\right) h$$

But the point(x_{i+1}, y_{i+1}) on RHS is still unknown. So we have to find it by extrapolation from (6.13)

$x_{i+1} = x_i + h$ and $y_{i+1} = y_i + hf(x_i, y_i)$

Substituting

$$Y_{i+1} = y_i + \left(\frac{f(x_i, y_i) + f(x_i + h, y_i + hf(x_i, y_i))}{2}\right) h$$

$hf(x_i, y_i) = k_1$

$hf(x_i + h, y_i + hf(x_i, y_i)) = k_2$

Substituting for k_1

$hf(x_i + h, y_i + k_1) = k_2$

$$y_{i+1} = y_i + \left(\frac{k_1 + k_2}{2}\right)$$

This is the second order R-K equation. In this k_1 is to predict the next term and k_2 is to correct the predicted term. So, this is a predictor-corrector method.

Python code for second order R-K - C.6.12

```
#RK Method second
def f(x,y):
    y=2*y/x
    return y
x= float(input('Initial value of x = '))
y= float(input('Initial value of y = '))
xf= float(input('Final value of x  = '))
h= float(input('Step size       = '))
print ('   x- Value          y- Value')
while x <= xf:
    print ('   %6.3f          %6.3f' %(x,y))
    if x + h > xf: break
    k1 = h*f(x,y)
    k2 = h*f(x+h,y+k1)
    y = y + (k1+k2)/2
    x = x + h
#print ('   %6.3f          %6.3f' %(x,y))
print (' Integral of the given function = %9.4f%y)
#end
```

Output

```
Initial value of x = 1
Initial value of y = 2
Final value of x  = 1.5
Step size       = 0.25
   x- Value          y- Value
```

1.000 2.000
1.250 3.100
1.500 4.443

Integral of the given function = 4.4433

Example 16:

$dy/dx = 2y/x$ is with an initial value y(1) = 2. Estimate y(1.5)with a step size 0.25

Answer:

Given f(x) = 2y/x , x=1, y=2

Step 1 y(1)=2 given

Step 2 $k_1 = hf(x_i, y_i) = 0.25(2 \times 2/1) = 1$

$k_2 = hf(x_i + h, y_i + k_1)$

$k_2 = 0.025 \times f(1 + 0.25\ ,2 + 1)$

$k_2 = 0.025\left(\frac{2 \times 3}{1.25}\right) = 1.2$

$Y_{i+1} = y_i + \left(\frac{k_1 + k_2}{2}\right)$

$Y_{1.25} = 2 + \left(\frac{1 + 1.2}{2}\right) = 3.1$

Step 3 $k_1 = 0.25 \times f(1.25,3.1) = 0.25 \times 4.96 = 1.24$

$k_2 = 0.025 \times f(1.5\ ,3.1 + 1.24) = 1.447$

$Y_{1.5} = 3.1 + \left(\frac{1.24 + 1.447}{2}\right) = 4.4435$

$\therefore y(1.5) = 4.447$

6.16 Taylor's series expansion

Tailor's series is one of the most useful series expressions of a function. In this method, expansion for y(x) around x=x_0

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + \frac{(x - x_0)^2}{2!}y''(x_0) + \cdots$$

Expansion for cos(x) about $x_0 = 0$

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + \frac{(x - x_0)^2}{2!}y''(x_0) + \cdots$$

$$cos(x) = Cos(0) - (x - 0)sin(0) - \frac{(x - 0)^2}{2!}cos(0) + \frac{(x - 0)^3}{3!}sin(0) + \frac{(x - 0)^4}{4!}cos(0) - \frac{(x - 0)^5}{5!}sin(0) - \frac{(x - 0)^6}{6!}cos(0) \ldots$$

$$cos(x) = 1 - 0 - \frac{x^2}{2!} + 0 + \frac{x^4}{4!} - 0 - \frac{x^6}{6!} \ldots$$

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

Python code for cos(x) - C.6.13

```
#Taylor series-cosine
from math import *
deg=float(input('Enter angle in degree '))
rad=deg*pi/180
cosx=1
for i in range (1,51):
    even=2*i
    cosx=cosx+((-1)**i)*(rad**even)/factorial(even)
print ('Cos', deg ,' = ',cosx)
#end
```

Output

Enter angle in degree 50
Cos 50.0 = 0.6427876096865393

Expansion for sin(x) about $x_0 = 0$

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + \frac{(x - x_0)^2}{2!}y''(x_0) + \cdots$$

$$sin(x) = sin(0) + (x - 0)cos(0) - \frac{(x - 0)^2}{2!}sin(0) - \frac{(x - 0)^3}{3!}cos(0) + \frac{(x - 0)^4}{4!}sin(0) + \frac{(x - 0)^5}{5!}cos(0) - \frac{(x - 0)^6}{6!}sin(0) \ldots$$

$$sin(x) = 0 + x - \frac{x^3}{3!} + 0 + \frac{x^5}{5!} - 0 - \frac{x^7}{7!} \cdots$$

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Python code for sin(x) - C.6.14

```
#Taylor series-sine
from math import *
deg=float(input('Angle in degree '))
rad=deg*pi/180
sinx=rad
cosx=1
for i in range (1,51):
  odd=2*i+1
  sinx=sinx+((-1)**i)*(rad**odd)/factorial(odd)
print ('Sin', deg,' = ',sinx)
#end
```

Output

Angle in degree 50

Sin 50.0 = 0.7660444431189778

Example 17:

Estimate cos(50) by numerical method

Answer

x=50 degree = 50 x 3.14/180 = 0.8722 radian

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

$$= 1 - \frac{(0.8722)^2}{2} + \frac{(0.8722)^4}{24} - \frac{(0.8722)^6}{720} + \ldots$$

$$= 1 - 0.3804 + 0.02411 - 0.000611 = 0.6431$$

$$\therefore \cos(50) = 0.6431$$

Example 18:

Estimate sin(50) by numerical method

Answer

x=50 degree = 50 x 3.14/180 = 0.8722 radian

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

$$\sin(50) = 0.8722 - \frac{0.8722^3}{6} + \frac{0.8722^5}{120} - \frac{0.8722^7}{5040} + \ldots$$

$$= 0.8722 - 0.1106 + 0.004206 - 0.000076 = 0.76573$$

$$\therefore \sin(50) = 0.76573$$

Exercise

One word type questions

1. In a difference table, $\Delta^2 y_1 =$ _____
2. In a difference table, $\Delta^2 y_1 - \Delta^2 y_0 =$ _____
3. A difference table can be used for _____ and _____
4. If the n^{th} difference operator is a constant, the interpolation formula will be accurate with----degree terms.
5. A table with given data points and higher-order differential operators is known as ----
6. Bisection method is a subdivision of _____ set of formulas.
7. From Newton's forward difference interpolation $\left[\frac{dy}{dx}\right]_{x=x_0} = \ldots$
8. The first-order R-K method is known as _____
9. To predict a value, if we are using the slope of four points, it is a ____ order R-K
10. Simpson's rule is accurate only if, the number of elements is ____

Short answer type questions

1. What is meant by the numerical method of solution?
2. Explain the advantages of the numerical method.
3. Explain the term 'curve fitting'.
4. What is an extrapolation?
5. Explain the forward differential operator.
6. Explain the backward differential operator.
7. Explain the central differential operator.
8. What is meant by the finite differential method?
9. Explain the Euler method.
10. Starting with the Euler method, explain the second-order R-K method.
11. Explain the R-K method to solve a first-order differential equation.
12. Why second-order R-K method is more accurate than the Euler method.
13. The second-order R-K method is called as a predictor-corrector method. Why?

Problems/Programmes

1. By Newton Raphson method find the solution of $x^2 - 2x - 1$
2. By Newton Raphson method find the solution of $\sin(x) - 2x + 1 = 0$
3. Find the equation of the best fit straight line for the following data points.

X	1	2	4	5	6	7	8
Y	-3.5	-1.4	0.8	5.2	7.4	9.6	11.8

4. Find the velocity at x=0.2 using the following table.

x	0.1	0.3	0.5	0.7	0.9
v	0.72	1.81	2.73	3.47	3.98

5. Find $\sqrt[4]{14}$
7. Find the equation of the straight-line, which is fittest for the following data.
8. Construct a difference table with the following data.

X	-2	-1	0	1	2
Y	-3.150	-1.300	0.620	2.880	5.378

9. The observed data in an experiment can be tabulated as follows. Estimate the first-order equation to satisfy the general behaviour of data.

X	-2	-1	0	1	2
Y	-7.3	-2.5	2.3	7.1	11.9

10. Compute y at x=2.5 by Newton's forward difference interpolation

X	1	2	3	4
Y	1	8	27	64

11. The following table represents tar and nicotine in popular brands of cigarettes. Fit a straight line with this data.

X	8.3	12.3	18.8	22.4	23.1	24
Y	0.32	0.46	1.10	1.32	1.26	1.44

12. Compute the value of y when x=323.5 from the following table.

X	321.0	322.8	324.6	326.4
Y	2.5065	2.5089	2.5108	2.5118

13. The values of stress corresponding to different values of strain, in a test are given below. Find the strain energy given by $\int \sigma de$

Strain(e)	0.25	0.30	0.35	0.40	0.45
Stress(σ)	24.1	25.5	26.6	27.3	27.9

14. A rocket is launched from the ground. The variation of velocity is as tabulated below. Find the acceleration at (1) 30 seconds (2) 35 seconds:

X	0	10	20	30	40	50	60	70

Y	30.	31.6	33.4	35.4	37.7	40.	43.2	46.6

15. Evaluate

$$\int_0^1 \frac{dx}{1+x^2}$$

16. From the following table compute the first derivative at x=-3

X	-3	-2	-1	0	1	2	3
Y	-33	-12	-3	0	3	12	33

17. Using the table of the above question find the first derivative at x=-1.5
18. Solve $dy/dx = x^2 + y^2$ at y(0.8). Given that y(0)=1
19. Evaluate

$$\int_{-1}^{1} e^x dx$$

20. Solve the differential equation $dy/dx = -y^2$ at y(2). Given that y(1)=1.
21. Find sin(35) and cos(45) by Taylor series expansion.

Long answer type questions

1. Explain the curve fitting by least square approximation. Develop a Python programme for it.
2. Explain the concept of a difference table with an example. What is its use?
3. Derive Newton's formula for interpolation. Develop a Python programme for it.
4. Explain the method of numerical integration for a given function. Develop a Python programme for it.
5. Explain the method of numerical differentiation when the point is not in the tabular form. Develop a Python programme for it.
6. Explain the Runge-Kutta methods for the solution of first-order differential equations. Develop a Python programme for it.
7. Explain the Taylor series expansion for sin x and cos x. Develop a Python programme for it.